Reliable RL: An Algorithmic Perspective

Logan Engstrom

(with A. Ilyas*, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry)
Reinforcement Learning (RL)
Reinforcement Learning (RL)
Reinforcement Learning (RL)
Is RL “real-world-ready”?
Is RL “real-world-ready”?

(Spoiler: No)

Deep RL is unreliable even in simple settings...
How do we get reliable RL?

An algorithmic understanding of modern RL methods
The RL Setup

Environment
Initial state $s_0$

Agent
Initial policy $\pi_0$
The RL Setup

Environment

State

Sampled action

Action distribution

Agent

Initial state $s_0$

Initial policy $\pi_0$
The RL Setup

Environment

Reward, State

Agent

Next state $s_t$

Initial policy $\pi_0$
The RL Setup

Environment

Reward, State

Agent

Next state $s_t$

Updated policy $\pi_t$
The RL Setup

Environment

Next state $s_t$

Reward, State

Sampled action

Action distribution

Agent

Updated policy $\pi_t$

Goal: Maximize expected total reward

(over trajectories)
Policy Gradient Algorithms
Policy Gradients

Key Principle: View our goal as an optimization problem

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{(s,a) \in \tau} r(s, a) \right]$$
Policy Gradients

Key Principle: View our goal as an optimization problem

\[ \theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{(s, a) \in \tau} r(s, a) \right] \]

- Expected value (over sampled trajectories) under current policy
- Total reward
Policy Gradients

**Key Principle:** View our goal as an optimization problem

\[ \theta^* = \arg \max_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{(s,a) \in \tau} r(s,a) \right] \]

Method of choice: gradient descent
Policy Gradients

Key Principle: View our goal as an optimization problem

$$\theta^* = \arg \max_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{(s,a) \in \tau} r(s, a) \right]$$

No gradient access

Method of choice: gradient descent
Policy Gradients

Can we instead get a good *estimate* of the gradient?

$$\nabla_{\theta} E_{\tau \sim \pi_{\theta}} \left[ \sum_{(s,a) \in \tau} r(s, a) \right] = ???$$
Policy Gradients

Can we instead get a good estimate of the gradient?

\[ \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{(s,a) \in \tau} r(s,a) \right] = \mathbb{E}_{\tau \sim \pi_\theta} \left[ g(\tau) \right] \]

The Policy Gradient
Policy Gradients

Can we instead get a good *estimate* of the gradient?

\[
\nabla_{\Theta} \mathbb{E}_{\tau \sim \pi_{\Theta}} \left[ \sum_{(s,a) \in \tau} r(s,a) \right] = \mathbb{E}_{\tau \sim \pi_{\Theta}} [g(\tau)]
\]

\[\approx \frac{1}{N} \sum_{\tau \sim \pi_{\Theta}} [g(\tau)]\]
Policy Gradients

Can we instead get a good estimate of the gradient?

\[ \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{(s,a) \in \tau} r(s,a) \right] = \mathbb{E}_{\tau \sim \pi_\theta} [g(\tau)] \approx \frac{1}{N} \sum_{\tau \sim \pi_\theta} [g(\tau)] \]

Then: use estimate in gradient descent!
Policy Gradient Successes
The Rotten Truth of Deep RL
The Rotten Truth of Deep RL

Deep RL can successfully solve tasks, but has...

- Poor reliability over repeated runs

[Henderson et al., 2017a,b] [Lewis et al., 2018]
The Rotten Truth of Deep RL

Deep RL can successfully solve tasks, but has...

- Poor reliability over repeated runs
- High sensitivity to hyperparameters

[Henderson et al, 2017a,b] [Lewis et al, 2018]
The Rotten Truth of Deep RL

Deep RL can successfully solve tasks, but has...

- Poor reliability over repeated runs
- High sensitivity to hyperparameters

[Henderson et al, 2017a,b] [Lewis et al, 2018]
The Rotten Truth of Deep RL

Deep RL can successfully solve tasks, but has...

- Poor reliability over repeated runs
- High sensitivity to hyperparameters
- Poor robustness to environmental artifacts

[Henderson et al, 2017a,b] [Lewis et al, 2018]
The Rotten Truth of Deep RL

Deep RL can successfully solve tasks, but has...

- Poor reliability over repeated runs
- High sensitivity to hyperparameters
- Poor robustness to environmental artifacts

Notably, benchmarks don’t reveal these problems
The Rotten Truth of Deep RL

Deep RL can successfully solve tasks, but has...

- Poor reliability over repeated runs
- High sensitivity to hyperparameters
- Poor robustness to environmental artifacts

Notably, benchmarks don’t reveal these problems

Where do such issues come from?

Hard to know: deep RL algorithms have many moving parts!
Implementation Obscures Deep RL Algorithms

Source: GitHub issues
Implementation Obscures Deep RL Algorithms

Source: GitHub issues
Implementation Obscures Deep RL Algorithms

Source: GitHub issues
Implementation Obscures Deep RL Algorithms

Deep RL algorithms are complicated & underspecified!

Source: GitHub issues
Implementation Obscures Deep RL Algorithms

- Without Optimization
- With Optimization

Maximum Reward

“Orthogonal” NN initialization
Implementation Obscures Deep RL Algorithms

- Without Optimization
- With Optimization

Maximum Reward

"Orthogonal" NN initialization
Implementation Obscures Deep RL Algorithms

- Reward Scaling
- LR Annealing
- Orthogonal init
- Value Clipping

With Optimization vs. Without Optimization
Implementation Obscures Deep RL Algorithms

Performance hugely varies with (seemingly) small changes!
Implementation Obscures Deep RL Algorithms

- Deep RL methods are complicated & underspecified
- Reasons for unreliability, performance are unclear
- Deep RL methods are poorly understood!
Back to First Principles
Back to First Principles

- Gradient Estimates
- Value Prediction
- Optimization Landscapes
- Trust Regions
Gradient Estimation

Key assumption of policy gradient framework:

$$\nabla_\theta \mathbb{E}_{\tau \sim \theta}[R(\theta)] \approx \frac{1}{N} \sum_{\tau \sim \theta} g(\tau)$$
Gradient Estimation

Key assumption of policy gradient framework:

$$\nabla_\theta \mathbb{E}_{\tau \sim \theta}[R(\theta)] \approx \frac{1}{N} \sum_{\tau \sim \theta} g(\tau)$$

How valid is this?
Gradient Estimation

$\theta_t$ (current policy parameters)
Gradient Estimation

$g_t^{(2)}$

$g_t^{(1)}$

$\theta_t$ (current policy parameters)
Gradient Estimation

\[ g_t^{(1)} = \frac{1}{k} \sum_{i=1}^{k} \ldots \]

(k-sample gradient estimate)

\[ \theta_t \text{ (current policy parameters)} \]
Gradient Estimation

$g_t^{(2)}$

$g_t^{(1)}$

$\theta_t$ (current policy parameters)
Gradient Estimation

\[ \theta_t \] (current policy parameters)
Gradient Estimation

Gradient Variance (mean pairwise correlation)

$\theta_t$ (current policy parameters)
Gradient Variance

![Graph showing gradient variance over the number of state-action pairs for TRPO, PPO, and PPO-M algorithms. The x-axis represents the number of state-action pairs, ranging from $100$ to $10^7$, and the y-axis represents the average pairwise cosine similarity. The lines for each algorithm show an increasing trend as the number of state-action pairs increases.](image-url)
Gradient Variance

- **Black line**: relevant sample regime
- Gradients are less concentrated than they could be
- Less correlated for "harder" tasks, later iterations
Gradient Estimation

- No good understanding of training dynamics
  - How does variance influence optimization?
  - Can we use insights from stochastic opt?
- Missing a link from reliability to sample size
Gradient Variance

[Graph showing average pairwise cosine similarity against number of state-action pairs for different algorithms: TRPO, PPO, and PPO-M.]
Gradient Estimation

- No good understanding of training dynamics
  - How does variance influence optimization?
  - Can we use insights from stochastic opt?
- Missing a link from reliability to sample size
Value Prediction

Gradient estimation is hindered by high variance!

Observation: If we can estimate the value of a state, can significantly lower variance

(The value of a state is the cumulative expected reward received after visiting the state)

Intuition: Need to separate action quality from state quality
Value Prediction

Variance reduction needs good value estimates

In Deep RL, values come from a neural network

To what degree do we actually reduce variance?
Value Prediction

![Graph showing average pairwise cos sim against # State-action pairs with Baselines: $V_0$, Zero, $V^*$ and # Iteration: 150.](image)
Value Prediction

Agent does significantly worse than optimal!
Value Prediction

- Might look small, but using a value network makes big difference
- How would using the true value affect training?
- Can we get better value estimates?

True value function
Agent's value function
No value function
Optimization Landscapes

Assumption: taking gradient steps increases reward

How valid is this assumption in practice?
Optimization Landscapes

Step 0

Reward

Random direction

Step direction
Optimization Landscapes
Optimization Landscapes

Step 0

$N(0, I^d)$

Reward

Random direction

agent step taken

Step direction
Optimization Landscapes

Step 0

Reward
(1000 trajectories)

$N(0, I^d)$

agent step taken

Reward

Random direction

Step direction

Step direction

Step direction
Optimization Landscapes

Step 0

Reward

agent step taken

Random direction

Step direction
Optimization Landscapes

Step 0

Step 150

Reward

Random direction

Step direction

agent step taken
Optimization Landscapes

Step 0

Step 300

Reward

Random direction

Step direction

agent step taken

Steps are often not predictive
Optimization Landscapes

Step 0

Step 300

Reward

Steps are often not predictive

What's going on here?
Optimization Landscapes

Methods iteratively maximize a “surrogate reward”

*(not the true reward!)*
Optimization Landscapes

Step 0

Step 300

Steps are often not predictive
Optimization Landscapes

Step 0

Step 300

Reward

Random direction

Step direction

agent step taken

Steps are often not predictive

What's going on here?
Optimization Landscapes

Methods iteratively maximize a "surrogate reward"

(not the true reward!)

How do surrogate rewards compare with true rewards?
Optimization Landscapes

Surrogate Landscape

- Reward
- Step direction
- Random direction
- Agent step taken
Optimization Landscapes

Surrogate Landscape

Reward Landscape

Reward

agent step taken
Optimization Landscapes

Surrogate Landscape

Reward Landscape

Step 150
Optimization Landscapes

Surrogate Landscape

Reward Landscape

Step 300
Optimization Landscapes

Surrogate Landscape

Reward Landscape

Step 450
Optimization Landscapes

Surrogate Landscape

Reward Landscape

Surrogate often misaligned with rewards!

Step 450
Optimization Landscapes

Surrogate Landscape

Reward Landscape

Step 450

agent step taken
Surrogate often misaligned with rewards!
Optimization Landscapes

All landscapes so far are in the high sample regime.

How do landscapes appear to the agent? (~20 trajectories)
Optimization Landscapes

20-sample estimates

Reward

agent step taken
Optimization Landscapes

20-sample estimates

Reward

agent step taken

20 trajectories per reward estimate
Optimization Landscapes

20-sample estimates

200-sample estimates

1000-sample estimates

Reward

agent step taken
Optimization Landscapes

20-sample estimates
200-sample estimates
1100-sample estimates

using many samples induces a smooth landscape...
Optimization Landscapes

20-sample estimates

200-sample estimates

1100-sample estimates

using many samples induces a smooth landscape...

... but improvement is hard to detect in the agent's sample regime
Optimization Landscapes

- Surrogate landscapes are often not reflective of rewards
- How can we better navigate the reward landscape?
Trust Regions

$\theta_t$
Trust Regions

\[ \theta_t \rightarrow \theta_{t+1} \]
Trust Regions
Trust Regions
Trust Regions
Trust Regions

Trust region
Trust Regions

Trust region
Trust Regions
Trust Regions

TRPO and PPO: Motivated by KL-based trust region:

$$\max_s D_{KL} \left( \pi_{\theta_{t+1}}(\cdot | s) \middle| \pi_{\theta_t}(\cdot | s) \right) \leq \delta$$

“keep the max distance between action distributions small”

But relax to an expectation:

$$\mathbb{E}_{s \sim \theta_t} \left[ D_{KL} \left( \pi_{\theta_{t+1}}(\cdot | s) \middle| \pi_{\theta_t}(\cdot | s) \right) \right] \leq \delta$$

“keep the mean distance between action distributions small”
Trust Regions

What happens in practice?
Trust Regions

- **TRPO** maintains trust region
- **PPO** algorithm does not!
- ... but **optimizations** help
Trust Regions

- What part of algorithms keep trust regions?
- How do we reason about algorithms when they use such loose relaxations?
- How can we capture different kinds of uncertainty in our trust regions?
Takeaways
Recap

- Deep RL methods are complicated
- Deep RL training dynamics are poorly understood
  - Steps are often uncorrelated
  - Surrogate rewards do not match true rewards
  - Trust regions do not hold
How do we proceed?

- Reconciling RL with our conceptual framework
  - How can we make algorithms better follow our conceptual framework?
- Rethinking primitives for modern settings
  - How do we deal with high dimensionality? Algorithm “optimizations?” Non-convex function approximators?
- Better evaluation for RL systems
  - Benchmarks don’t capture reliability, safety, or robustness of RL agents
Optimization Landscapes

Surrogate Landscape

Reward Landscape

Step 450